Matlab m-file for Digitizing Graphs

Harald E. Krogstad, NTNU, Trondheim

The following simple m-file helps digitizing graphs and drawings by picking a series of points from the graph using the mouse pointer provided by the Matlab function `ginput` (see the Matlab documentation). The program requires that the user specifies the x-axis and the y-axis by defining two different points on each. The axes may be chosen at will and need not cross in a particular point, e.g. the origin. The code is also able to handle logarithmic axes.

There are two different coordinate systems to consider:

1. The coordinate system given on the graph itself, which is also the system we want to relate our data points to
2. The coordinate system of the image of the graph (the pixel numbering used by Matlab)

The output from the routine is an $N \times 2$ array of data points $(x_i, y_i)_{i=1}^N$ given in the graph coordinate system.

If the graph only exists on paper, it first needs to be scanned in a graphical format Matlab understands, e.g. the `bmp`, `jpg` or `png` formats (See the Matlab documentation for a survey of other available formats).

With the image of the graph available, the first step then consists of getting the graph into Matlab, and this is done by a call to the Matlab function `imread`, e.g.

```
A = imread('Figure.png','png');
```

The image is then displayed by calling `image`,

```
image(A);
```

At this stage it may be suitable to zoom and scale the image to a size which is easy to digitize accurately.

The digitizing function is then called (for the first time without any input argument):

```
[data,calib] = digitization([calibin])
```

Here `calib` and `calibin` are Matlab structures that contain the reference calibration points on each axis if it turns out necessary to carry out the digitization in separate sessions.

Note: If the graph already exists in the form of a Matlab (vector graphics) fig-file, data values may be recovered directly by identifying the line graphical elements making up the graph (See Matlab documentation for further information). As an example, for the *current graph*, the lines are identified by:

```
lines = get(gca,'children')
```

followed by

```
xdat = get(lines(1),'xdata');
ydat = get(lines(1),'ydata');
```

... and so on for all lines on the graph.

1 The Algorithm

The digitalization algorithm does not require that the axes on the image are completely orthogonal, or exactly vertical/horizontal. The coordinates $\mathbf{x}_c$ and $\mathbf{y}_c$ are obtained from the data point by skew projections, as illustrated on Figure 1. This assumes that the distortion is uniform over the whole drawing, which should be reasonable, at least for scanned images. The points $(\mathbf{x}_0, \mathbf{x}_1)$ and $(\mathbf{y}_0, \mathbf{y}_1)$ are user-defined points on the x- and y-axis with given image as well as real coordinates. The points should be selected at some distance so the axes are accurately defined.

By denoting the data point $\mathbf{x}_d$, we obtain the following equations in the 4 variables (t, s, α, β):

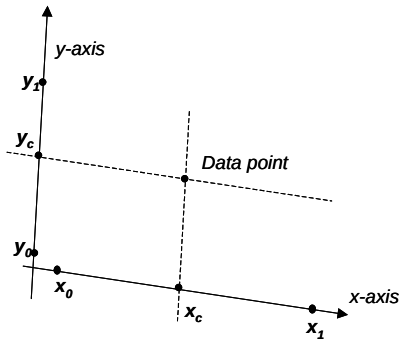

Figure 1: The graph illustrates how the x and y coordinates for the data point, $\mathbf{x}_d$ are identified, assuming straight axes and a pair of points with known locations on both.

$$\begin{aligned} \mathbf{x}_c &= \mathbf{x}_0 + t\,(\mathbf{x}_1 - \mathbf{x}_0)\,, \\ \mathbf{y}_c &= \mathbf{y}_0 + s\,(\mathbf{y}_1 - \mathbf{y}_0)\,, \\ \mathbf{y}_c &= \mathbf{x}_d - \alpha\,(\mathbf{x}_1 - \mathbf{x}_0)\,, \\ \mathbf{x}_c &= \mathbf{x}_d - \beta\,(\mathbf{y}_1 - \mathbf{y}_0)\,, \end{aligned} \tag{1}$$

or

$$\begin{aligned} \mathbf{x}_0 + t\,(\mathbf{x}_1 - \mathbf{x}_0) &= \mathbf{x}_d - \beta\,(\mathbf{y}_1 - \mathbf{y}_0)\,, \\ \mathbf{x}_d - \alpha\,(\mathbf{x}_1 - \mathbf{x}_0) &= \mathbf{y}_0 + s\,(\mathbf{y}_1 - \mathbf{y}_0)\,. \end{aligned} \tag{2}$$

The constants t and s are what is needed to obtain the actual coordinates of the data points. E.g., the x-coordinates,

$$x_c = x_0 + t\,(x_1 - x_0)\,, \tag{3}$$

for a linear axis, and

$$x_c = \exp\left[\log x_0 + t\,(\log x_1 - \log x_0)\right] \tag{4}$$

for a logarithmic axis.

The equations may be written as the matrix system

$$M \begin{bmatrix} t & \alpha \\ \beta & s \end{bmatrix} = R, \tag{5}$$

where

$$\begin{aligned} M &= \left[(\mathbf{x}_1 - \mathbf{x}_0)\,(\mathbf{y}_1 - \mathbf{y}_0)\right], \\ R &= \left[(\mathbf{X}_d - \mathbf{x}_0)\,(\mathbf{X}_d - \mathbf{y}_0)\right]. \end{aligned} \tag{6}$$

It is convenient to introduce corresponding row vectors for (t, s, α, β) and solve

$$M \begin{bmatrix} T & A \\ B & S \end{bmatrix} = R. \tag{7}$$

The m-file `digitization.m` implements the algorithm and also contains an interactive section for defining reference points on the axes.

```
function [data,calib] = digitization(calibin)
% FUNCTION FOR DIGITIZING GRAPHS.
% Axes reference information may be predefined in the structure
% calibin, or it may be created, in which case the user is
% inquired about the following input:
%
% Two points on the x-axis and two points on the y-axis have
% to be chosen, preferably at some distance so as to define
% the axis accurately.
%
% Axes do not need to cross at a particular point and both
% linear and logarithmic axes are accepted.
%
% Harald E. Krogstad, NTNU, 2006
%-----------------------------------------------------------------
if nargin == 0
    %
    % Dialog for defining the x-axis
    x0  =  input('Enter x coordinate for x0: ');
    disp('Point to x0!')
    x0p = (ginput(1))';
    %
    x1 =  input('Enter x coordinate for x1: ');
    disp('Point to x1!')
    x1p = (ginput(1))';
    linlogx = input('Linear/logarithmic x-axis? (0=lin.,1=log.):');
    %
    % Dialog for defining the y-axis
    y0  =  input('Enter y coordinate for y0: ');
    disp('Point to y0!')
    y0p = (ginput(1))';
    %
    y1 =  input('Enter y coordinate for y1: ');
    disp('Point to y1!')
    y1p = (ginput(1))';
    linlogy = input('Linear/logarithmic y-axis? (0=lin.,1=log.):');
    %
    % Collect axis reference structure
    calib.x0   = x0;     calib.x1 = x1;
    calib.y0   = y0;     calib.y1 = y1 ;
    calib.x0p  = x0p;   calib.x1p = x1p;
    calib.y0p  = y0p;   calib.y1p = y1p;
    calib.linlogx = linlogx ;
    calib.linlogy = linlogy ;
else
    %
    % Unpack existing axis reference structure
    calib = calibin;
    x0  = calib.x0;    x1 = calib.x1;
    y0  = calib.y0;    y1 = calib.y1;
    x0p = calib.x0p;  x1p = calib.x1p;
    y0p = calib.y0p;  y1p = calib.y1p;
    linlogx = calib.linlogx;
    linlogy = calib.linlogy;
end
%
% Get data-points
disp('Pick data points, - end with Enter: ')
Xp = ginput;
%
% Find picture coordinates
Ndata = size(Xp(:,1));
Mm = [ (x1p - x0p)   (y1p - y0p) ]^(-1);
Xpmx0 = Xp' - x0p*ones(1,Ndata);
```

```
Xpmy0 = Xp' - y0p*ones(1,Ndata);
T = Xpmx0'*Mm(1,:)' ;
S = Xpmy0'*Mm(2,:)' ;
%
% Transform to graph (actual) coordinates
if linlogx == 1
   % Log x-axis
   xdata = exp( log(x0) +  (log(x1) - log(x0))*T ) ;
else
   % Linear x-axis
   xdata = x0 +  (x1 - x0)*T;
end
%
if linlogy == 1
   % Log y-axis
   ydata = exp( log(y0) +  (log(y1) - log(y0))*S ) ;
else
   % Linear y-axis
   ydata = y0 +  (y1 - y0)*S;
end
data = [xdata ydata];
```